IEEE Std 1609.2-2016 Guidance Note 6: Canonicalization of certificates

William Whyte, Security Innovation

Version 0.1, 2016-08-12

1 Introduction

This note identifies an internal inconsistency in IEEE Std 1609.2-2016 that has the potential to lead to lack of interoperability between different implementations, and gives guidance to implementers and certification testers as to the recommended interpretation.

The defect has a general and a specific case.

The general case is that all certificates input to any hash function in 1609.2 are intended to be "canonicalized", but in some parts of the standard this is stated explicitly and in other parts it is not stated explicitly. This should be addressed so the standard is clear and consistent in stating that certificates are canonicalized.

The specific case concerns input to the hash function when hashing an implicit certificate to recover the public key from the reconstruction value. Here, one of two certificates is explicitly noted as canonicalized while the other is not. This gives the impression that the second certificate is not to be canonicalized even though the intent of the standard is that all certificates are to be. An implementation that does not canonicalize the certificate will be incompatible (in the sense of deriving a different public key) from one that does, so implementers who differ on when certificates should be canonicalized will produce incompatible implementations.

This guidance note is structured as follows. Section 2 provides a summary of the issue, the impact, and current practice. Section 3 provides recommended changes to eliminate any ambiguity from the standard.

2 Summary of issue

Background. In 1609.2, elliptic curve points may be represented as "compressed", "uncompressed", or "x-only". Elliptic curve points may appear in certificates as public keys, as a part of the signature component r, or as reconstruction values. There are speed advantages to working with uncompressed points, but bandwidth advantages to working with compressed points, so the 1609.2 design allows data structures to be transmitted with elliptic curve points either compressed or uncompressed to best fit the particular size/performance tradeoff that applies at a particular time.

However, the hash of the certificate is used in a number of settings – when the certificate itself is signed and/or verified in the case of explicit certificates; when the implicit certificate is generated or the private key is recovered from it; and when the certificate is used to sign a message. In this case, the hash should be consistent no matter what the transmitted representation of the certificate is.

1609.2 Clause 6.4.3 therefore defines a "canonical" representation of the certificate as one in which all elliptic curve points are in "compressed" form except for the value r in the signature, which is represented in x-only form.

Issue: Although clause 6.4.3 says "When a certificate is encoded for hashing, it is canonicalized as follows..." not every location in 1609.2 that refers to hashing a certificate explicitly states that it is canonicalized. This could lead an implementer who wasn't paying close enough attention to hash certificates without canonicalizing them.

In particular, in section 6.4.8, the standard describes the input to reconstructing the public key from the reconstruction value in an implicit certificate. (The exact text is quoted below). This input is the concatenation of two hashes of certificates. The first certificate is noted as being canonicalized, the second is not. This could lead a reader to believe that the intent is that the second certificate is not canonicalized. In fact, the intent of the standard is that all certificates are canonicalized whenever they are hashed. The text in 6.4.8 is therefore potentially misleading in a significant way.

Current practice: We are not aware of any implementation that does not canonicalize the certificates and we are aware of multiple implementations that do canonicalize them. This guidance note is therefore, as far as we are aware, purely precautionary.

Impact: An implementation that does not canonicalize the certificate may calculate a different hash on a certificate from an implementation that does canonicalize. Since hashes are used in signature calculation and verification and in public key reconstruction, the two implementations may produce signatures that cannot be verified by each other or implicit certificates from which the public key cannot be correctly extracted.

Proposed resolution: Clarify 1609.2 throughout to make it clear that certificates are canonicalized.

3 Recommended interpretation

The following are the subclauses in 1609.2 where a certificate hash is referred to, and recommended changes.

5.3.1, Signature Algorithms:

Current text: step c)2)i): "*signer identifier input* shall be the certificate with which the message is to be verified."

Recommended text: "the certificate with which the message is to be verified, canonicalized as specified in 6.4.3".

5.3.2, Implicit Certificates:

Current text: "The encoded data input to the hash function is Hash (ToBeSignedCertificate from the subordinate certificate as specified in 6.4.8) || Hash (Entirety of issuer certificate as specified in 6.4.3

Recommended text: "The encoded data input to the hash function is Hash (ToBeSignedCertificate from the subordinate certificate as specified in 6.4.8, canonicalized as specified in 6.4.3) || Hash (Entirety of issuer certificate, canonicalized as specified in 6.4.3)

5.3.5, Public Key Encryption:

Current text: in discussing the input to the encryption parameter P1, says "The certificate shall be put in canonicalized form when hashing: see 6.4.3."

Recommended text: the current text does not need to be changed.

6.3.4, SignedData:

Current text: "*Signer identifier input* equal to the COER-encoding of the `Certificate` that is to be used to verify the SPDU, canonicalized according to the encoding considerations given in 6.4.3."

Recommended text: the current text does not need to be changed.

6.4.3, CertificateBase: Defines canonicalization, no change necessary.

6.4.8, ToBeSignedCertificate:

This is the material referred to in section 2 above where one certificate is explicitly noted as being canonicalized and the other one isn't, potentially giving the impression that the second certificate is not meant to be canonicalized. The offending text is underlined below:

Current text:

For both implicit and explicit certificates, when the certificate is hashed to create or

recover the public key (in the case of an implicit certificate) or to generate or verify the signature (in the case of an explicit certificate), the hash is Hash (*Data input*) || Hash (*Signer identifier input*), where:

- *Data input* is the COER encoding of `toBeSigned`, canonicalized as described above.
- *Signer identifier input* depends on the verification type, which in turn depends on the choice indicated by `issuer`. If the choice indicated by `issuer` is `self`, the verification type is *self- signed* and the *signer identifier input* is the empty string. If the choice indicated by `issuer` is not `self`, the verification type is *certificate* and the *signer identifier input* is the COER encoding of the canonicalization per 6.4.3 of the certificate indicated by `issuer`.

In other words, for implicit certificates, the value H (CertU) in SEC 4, section 3, is for purposes of this standard taken to be H [H (canonicalized ToBeSignedCertificate from the subordinate certificate) || H (entirety of issuer Certificate)]. See 5.3.2 for further discussion.

Recommended text:

As above **except** that the last paragraph should be amended as follows (underlining added for emphasis and will not appear in the final text)

In other words, for implicit certificates, the value H (CertU) in SEC 4, section 3, is for purposes of this standard taken to be H [H (canonicalized ToBeSignedCertificate from the subordinate certificate) || H (canonicalized entirety of issuer Certificate)]. See 5.3.2 for further discussion.